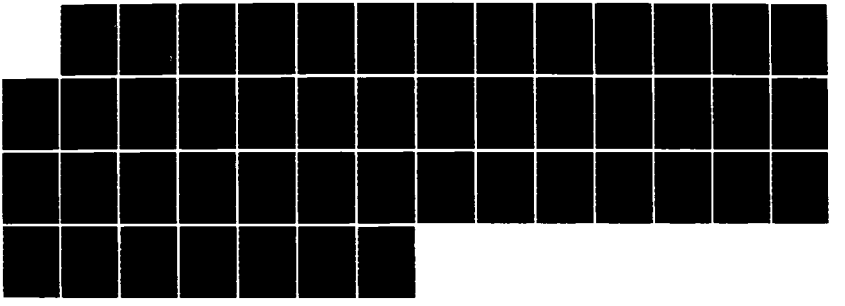
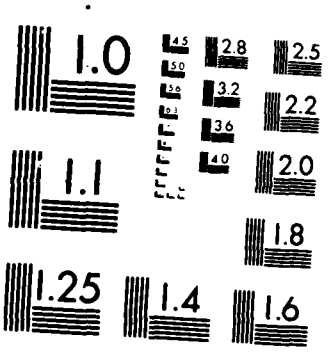
AD-A173 431 FUNCTIONAL TESTING OF LSI/VLSI (VERY LARGE SCALE INTEGRATION) BASED SYSTE..(U) STATE UNIV OF NEW YORK AT BINGHAMTON DEPT OF COMPUTER SCIENCE.. S Y SU JUN 83 1/1

UNCLASSIFIED DAAB07-82-J-J056 F/G 9/5 NL





# FUNCTIONAL TESTING OF LSI/VLSI BASED SYSTEMS WITH MEASURE OF FAULT COVERAGE

Final Report

June 1983

By: Stephen Y. H. Su, Project Director

Department of Computer Science
State University of New York

Prepared for:

Headquarters, US Army Communications-Electronics Command
Fort Monmouth, New Jersey 07703

Contract No. DAAB07-82-J-J05-6



The Research Foundation
State University of New York
Binghamton, New York 13901

TABLE OF CONTENTS


| | | PAGE |
|---|---|---|
| Abstract | | i |
| Chapter 1 | Overview of Recent Developments in Functional Testing of LSI/VLSI Devices | 1 |
| | I Introduction | 1 |
| | II Types of LSI/VLSI Chips and Their Testing | 3 |
| | III The State of the Art of Testing | 5 |
| | IV Fault-Models and Fault Coverage Measures | 6 |
| | V Current Functionsl Testing Approaches | 7 |
| | VI Comparisons Among Different Approaches | 13 |
| | VII Current Trends | 14 |
| References | | 16 |
| Chapter 2 | New Algorithms for Testing Instruction Decoding Function of Microprocessors | 18 |
| | I Introduction | 18 |
| | II Key Ideas | 20 |
| | III Assumptions and Notation | 22 |
| | IV Theorems and Algorithms | 27 |
| | V Complexity of the Algorithms | 34 |
| | VI Symmetric Fault Model | 38 |
| | VII Conclusions | 39 |
| References | | 40 |
| Appendix | | 41 |

## ABSTRACT

Due to the advances in the integrated circuit (IC) technology, more and more components are being fabricated into a tiny IC chip. Since the number of pins on each chip is limited by the physical size of the chip, the problem of testing becomes more difficult than ever, especially in the VLSI (Very Large Scale Integration) chips. This problem is aggravated by the fact that, in nearly all cases, integrated circuit manufacturers are not willing to release the detailed circuit diagram of the IC chip to the users. Yet, as users of the IC chips, to make sure that the implemented system is reliable, we need to test the IC chips and the systems made of the interconnection of these chips. The purpose of this project is to find efficient algorithms for testing LSI/VLSI chips and LSI/VLSI-based systems.

This report is organized into two chapters. Chapter 1 presents the state-of-the-art for the functional testing of LSI/VLSI devices with special emphasis on microprocessor testing. Various types of IC chips are briefly discussed. Different approaches for testing the functional faults of LSI/VLSI are surveyed and the comparison of these methods are given. Fault models for representing the faults and fault coverage of the tests are discussed. Some of the important unsolved problems and current trends in testing VLSI are pointed out.

Chapter 2 reports our new research results. We present three algorithms to test the instruction decoding function of microprocessors. The algorithms are based on the knowledge of some timing and control information available to users through microprocessor manuals and data sheets. The tests are functional in nature. We establish the order of complexity of the algorithms presented in this report. As an example, the test complexity for a microprocessor is computed and the results are compared with a known algorithm.

CHAPTER 1

# Overview of Recent Development in Functional Testing of LSI/VLSI Devices

## I. INTRODUCTION

The rapid growth of the integrated circuit technology has allowed LSI/VLSI devices to provide more powerful functions and to become more and more popular in various kinds of commercial, industrial and military applications. These complex devices need to be tested for correctness and guaranteed reliability before being adopted in an individual application. What makes the situations even harder than before is that due to the increasing complexity during the design and manufacturing process of such chips, the testing problem becomes much more difficult to handle. In view of the high complexity of VLSI, classical strategy for gate-level stuck-type testing can hardly fit well. The need for both manufacturer and user to find reliable, low-cost, good testing techniques has already become a major problem in modern technology.

Although the manufacturers have detailed knowledge of the logical and parametric behavior of the chips produced, they usually do not apply comprehensive, thorough testing to their products mainly due to economic reasons. To assure reliable operations, additional testing needs to be performed by the users. Users may have as many as three different testing environments -- ad hoc functional testing by a simple test circuit, assembling production test, and occasional field testing. As a matter of fact, due to the increasing number of LSI/VLSI users, it is becoming evident and more important that we develop good test generation algorithms to test these devices based on whatever information is available to users in manufacturers' data books and application notes.

In general, we should consider not only testing the conventional self-contained single chip devices (e.g., single chip microprocessor), but also testing other special-purpose devices (e.g., bit-sliced microprocessor), especially in military applications [16].

It is likely, with the evolutionary development of functions contained in newly produced chips, that much more complex and sophisticated test strategy will be needed in addition to whatever built-in testing hardware in the ships for increasing the testability.

During recent years certain kinds of techniques have been proposed in testing LSI/VLSI devices. They all have attempted to derive comprehensive functional tests using the limited information to users. To deal with the complexity of VLSI circuits in a comprehensive fashion, these approaches have used abstract, formal descriptions of VLSI behavior different from traditional test approaches.

Some testing approaches developed so far are based on the utilization of the concept of register transfer language (RTL) to describe the behavior of the devices. With the advent of the more advanced VLSI devices (e.g., the 4-th generation microprocessors) in which certain RTL operations in the system may be hidden from the user, it is obvious that either the existing test methods should be modified or different but similar strategies must be exploited.

In this chapter we attempt to summarize the current state-of-the-art on the development of functional test techniques for LSI/VLSI devices with special emphasis on the important area of microprocessor testing. The second section briefly discusses the types of LSI/VLSI chips from a general system organizational point of view and their individual testing approaches. Section three states the general state of the art in functional testing strategies.

Section four describes fault models and coverage measures. The effectiveness of a test approach is evaluated based on the percentage of fault covered by the tests it can achieve. Section five summarizes major functional testing approaches in the current literature and a simple comparison is made among them in Section six. The last Section includes some work which remains to be explored in this field and some future trends.

## II. TYPES OF LSI/VLSI CHIPS AND THEIR TESTING

In general, the LSI/VLSI chips available to users can be categorized from the system organizational point of view as the following four types:

- Microprocessors
- Memory chips
- Peripheral supporting chips
- Special-purpose functional chips

Microprocessors are usually utilized as the processing control unit of application systems and usually have the highest functional complexity among those four types. Peripheral supporting chips include those chips which perform interfaces and inter-module communications. They usually have limited specific functions imbedded within themselves and can be activated under program control. The memory chips are used only for program or data storage purpose; therefore, they usually receive the least emphasis in functional testing due to their low functional complexity. A lot of work has been done in the area of static functional testing of LSI semiconductor memories [18]. The special-purpose functional chips include those devices designed for speedy execution of certain functions (e.g., hardware multiplier, fast fourier transform and so on).

As we mentioned earlier, users must assure the reliable operations of purchased LSI/VLSI chips. This usually left them with several uncomfortable

alternatives. One simple alternative people often adopted in testing microprocessors is to purchase a set of test vectors from the vendor and use them to test the purchased chips. However, such ad hoc analyses usually have only been made on the chips and therefore are not made on the basis of a theoretical model with provable comprehensiveness. Another possible approach is to run a pseudorandom test sequence of test vectors for simplicity and minimum cost; but, to derive reliable measures of fault coverage using this method is a nondeterministic problem. Actually, the reasonable way to test microprocessors is to use those techniques attempting to derive high fault coverage functional test programs using the limited information available to users (e.g., those contained in user's manuals). Formal, abstract descriptions of processor behaviors are used to enable the proof of comprehensiveness.

To attack the high functional complexity of microprocessors, modular decomposition techniques are usually used to subdivide the microprocessor into functional modules whose behaviors could be individually verified using appropriate test procedures. Two situations may often be observed in testing microprocessors:

1) Testing is focused on data path functions. The fault types of control path is usually simplified.
2) Modules and tests are developed on an ad hoc basis. [7]

The order of test sequence is of major concern during the testing of microprocessors. Basically, a partial or total order is looked for to determine one way appropriate for testing the instructions. This order is usually derived from the relations of dominance and associated parameters which may be structural or functional. Two well-known test organizations can be applied: the start small and the start big method. The start small method tests only a

small portion of hardware and then uses the tested part to detect the fault in other parts. Each additional test adds a small quantity of hardware to the previously tested parts until all are tested. This method requires an ordering among the set of instructions with respect to testing. The start big test starts with the verification of the whole system to determine which part is faulty and proceeds to sequentially narrow down the region in which the fault occurs.

The testing of memory chips only concerns with read and write processes of the individual storage elements and can usually be done by various memory testing strategies [18]. Further advanced testing of real world memory faults (e.g., intermittent and data dependent faults) are still worthwhile in practical applications which require high reliability.

## III. THE STATE OF THE ART OF TESTING

There are mainly two trends in the test methods for LSI/VLSI devices. The first one is functional testing which generally views the chip-under-test as a system. For example, if a microprocessor is being tested, the knowledge of the functional block diagram allows the microprocessor to be divided into physical blocks such as arithmetic logic unit, control unit and so on. Each block is characterized by its function. Testing a microprocessor then consists of exercising every block with specified test data in a given order. In general, test data can be one of the following:

1) Generated in random order.
2) Generated in a deterministic manner, usually a model is set up based on a high level functional description.
3) Exhaustively generated.

The second method is structural testing which assumes (as its name

suggests) that a knowledge of the detailed structure of the circuit is clearly known. The difficulties which arise when applying this kind of test method are:

- The lack of knowledge about the physical defects which can be present in the new technologies, and
- Due to the complexity of VLSI, it is very complex to enumerate all faults and to generate the tests for them.
- In practice, there is only a small possibility for a component user to know the detailed structure of the device-under-use, especially for LSI/VLSI devices.

A good test approach usually considers both functional and structural failures. Functional testing can be performed while taking into account the knowledge of the structural information to obtain enhanced test efficiency.

## IV FAULT MODELS AND FAULT COVERAGE MEASURES

To simplify the evaluation of the fault coverage in digital systems, a comprehensive fault model is usually set up and then the fault coverage is evaluated based on this model. One interesting way to set up the functional level fault model of digital devices other than microprocessor (e.g., VLSI, PLA) is the fault characterization technique [14]. The key idea is to derive the functional level fault model by simulating physical failures at the circuit level. Although this approach has primarily two limitations of being implementation dependent and being hard to attack complex VLSI modules, the latter problem can be solved by employing a two-step method simulating the large module based on small modules and multi-valued algebra [14].

There are two kinds of fault models for the testing of microprocessors. One is called the universal fault model which takes all possible faults into account and no faults are specific to any functions (but may be specific to

instructions/micro-instructions). The other is called functional fault model which partitions faults into several types according to different functions. The microprocessor testing approach [12] based on microprogramming concept uses the universal fault model, whereas the graph theoretic method [2] adopts the functional fault model.

In general, fault coverage is defined as the ratio of faults detected to the set of all faults considered. For simplicity it is also defined as the fraction of all singly-occuring faults which can be detected by a given set of test vectors for a specific device. For simplification, only the permanent static faults are considered in most fault measures in current testing techniques [9]. The evaluation of fault coverage data is usually done by simulation through software. Essentially, the simulator models the responses of n+1 circuits: the fault-free circuit and the n single-fault circuits. The fault simulator results are often only approximations to those actually observed during actual circuit testing. This is true even if the simulator modelled all reasonable types of faults.

## V. CURRENT FUNCTIONAL TESTING APPROACHES

In this section, first of all, we describe three major approaches for functional testing of microprocessors [1,2,3,4,12,13].

### (a) Graph Theoretic Approaches

There are two different techniques proposed in this category. In [1], each instruction of a microprocessor is represented by an "abstract execution graph" in which memory elements, including source and destination registers, are represented as circle-shape type-1 nodes, and the microoperations performed by the instruction are represented as square-shape type-2 nodes. An example of "ADDA n,X" from the MC6800 microprocessor is given in Fig 1. This instruction

first computes the effective address by adding n to X, using this effective address for fetching the operand. Then the operand is added to register A and store the sum in register A.

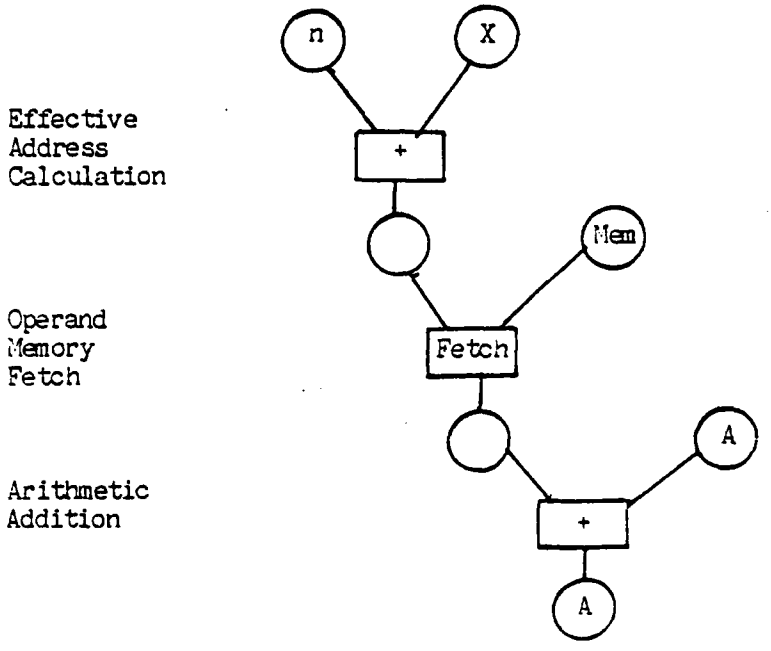



Figure 1

Two strategies, both start small and start big strategy can be used in generating test procedures. The start small testing is implemented by testing instructions from $c_i$ before instructions from class $c_i+1$, where i denotes the order of the class. To make the testing automated and complete, two stages of test are involved: first, verifying the structure of the abstract execution graph, and then verifying the correct function of each node. Memory elements are checked by running appropriate test patterns and every microoperation is checked by exhaustive testing.

A different graph theoretic approach, [2], proposed another method which makes use of register transfer level description of microprocessors. Instructions are classified into three types: transfer (T), manipulation (M), and branch (B). A directed s-graph is derived based on the basic functional, structural information and instructions set in which a node represents a register and each edge represents execution of instruction. An example of s-graph based on a part of a hypothetical microprocessor is given below and the corresponding graph is shown in Fig. 2.

$I_4$: add the contents of $R_1$ and $R_2$ and store the results in $_R$ (M-class)

$I_9$: jump instruction (B-class)

$I_7$: store $R_1$ into the main memory using implied addressing (T-class)

$R_6$: program counter

In, out: external memory at input/output devices

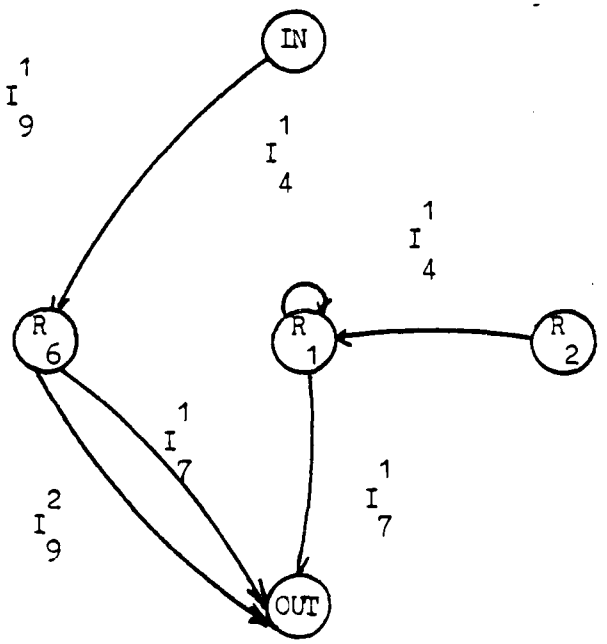



Figure 2

The microprocessor functions considered in this method are register decoding, instruction decoding and control, data storage, data transfer, and data manipulation. The faults specified for data storage and transfer functions are conventional stuck-type faults. A valid test corresponds to the execution of a sequence of instructions which transfer complementary data patterns from the input port to the output port. The faults for register and instruction decoding are failures to address the correct register or to execute the correct instruction. Tests for these faults involve tracing the sequence of instructions from input to output ports.

(b) Register Transfer Language (RTL) Technique

Using RTL, the behavior of a microprocessor is comprehensively described, and functional faults derived from them can be studied. In [4], two approaches for functional testing are given based on the RTL description. The first approach constructs a data graph from the RTL description and uses the existing algorithm such as the D-algorithm or path sensitizing method to generate the tests for functional faults. In the second approach, the symbolic simulations technique is used to generate tests for detecting faults in the control signals. In [5], a formal definition of RTL is defined as:

$$k:\ (t,c)\ R_d \leftarrow f(R_{s1},R_{s2},\ldots,R_{sv}),\ \rightarrow n$$

where,

k is the statement label

t is the timing and c is the condition to execute the statement

$R_d$ is the destination register

$R_{si}$ is the ith source register

f is an operation on $R_{si}$

← represents data transfer

→ n represents a jump to statement n

For example, the following RTL statement No. 17: $(T_5C_8)$ $R_2 \leftarrow R_3+R_5 \rightarrow 38$ means that when $T_5 = C_8 = 1$, the sum of $R_3$ and $R_5$ will be stored in $R_7$ and then the program jumps to statement No. 38.

Based on the above notation, eight categories of fault can then be identified as timing faults (t/t'), condition faults (c/c'), register decoding faults ($R_i/R_i$), instruction decoding (function selection) faults (f/f'), control faults (n/n'), data storage faults (($R_i$)/($R_i$)), data transfer faults (←/→') and data manipulation (function execution) faults ((f)/(f')). This set is functional comprehensive because the behavior of a CPU can be described by a sequence of RTL statement. Three procedures for testing those five fault categories (except timing, condition, and control faults) are derived. The testing requires the creation of executable sequences to form a "sensitizing" path which leads from a faulty statement to a statement producing faulty output information. The RTL technique seems to be a promising approach for functional testing. We will discuss this more in the next section.

(c) The Microprogram-Oriented Technique

This technique functionally describes a microprocessor as a set of microprograms derived from user available information [12]. The basic information required includes:

1) The set of internal registers and functional units.
2) The set of instructions and asynchronous control signals.
3) The behavior of signals at external pins and internal operations at each clock semicycle which can generally be derived from the timing charts in the user manuals.

Instruction cardinality is defined as the number of independent (subsequent) access to registers. In [12], it is shown that a necessary condition for the nonambiguous error coverage of a particular, not directly observable, instruction is the existence of a testing sequence of instructions with lower cardinality. A complete test requires that all instructions with all addressing modes be included in the test sequences. The microprogram-oriented approach is attractive because micro-instructions are basic to the operation of the control unit and nearly all of the current new microprocessors are basically microprogram controlled. Since the instruction decoder outputs a starting address to the micro control store which subsequently performs a set of micro-instructions corresponding to the incoming instruction, the length of the starting address is a critical factor because it eventually determines the efficiency of the final test sequence. If one can generate all the possible combinations of the starting address patterns, all instructions executions will be fully exercised. The sharing of micro-instructions among different instructions evidently tends to reduce the number of different starting addresses.

The above three testing approaches are proposed for testing microprocessors. Next, as for large scale digital devices, the extension of the D-algorithm suitable for functional level testing has been proposed [13,15]. General computer hardware description language constructs (e.g., if-then-else, case) and operators (e.g., addition) have been used to describe the behavior of the general logic network. Fault model similar to stuck-type will be used and the classical stuck-type testing technique can still be applied without major changes. By building the basic functional blocks from elementary components, it seems that modular composition and recursive extension can easily be applied

during the procedure of testing which in turn may provide great flexibility in adjusting the test strategy.

Memory devices have been tested for pattern sensitive faults [18]. The functional testing of memory from the user's point of view is to apply a sequence of "read" and "write" to check the storage function of each memory element.

## VI COMPARISONS AMONG DIFFERENT APPROACHES

Each of the three different approaches in testing microprocessors described in the previous section has its own individual advantages and shortcomings. A brief comment on each is given below.

Several observations can be made about the method in [1]. First, its coverage rests on the exhaustive testing of particular functions and memory elements. Hence it is an ad hoc method and will be specific to the devices-under-test (e.g., MC6800). Moreover, it presumes a black box view of the microprocessor in which each function is tested independently. These factors lead to the tests of excessive length. But they have made a useful contribution by devising a simple, rational scheme for ordering instructions according to easiness level of controllability and observability.

The graph approach in [2] is attractive for several reasons: First, it is based on the minimum, available information about the microprocessor (e.g., instruction and register sets). Second, microprocessor operations are decomposed into a set of functions. For each function, a specific single fault model is defined and a comprehensive test is rigorously derived. Third, the fault coverage was verified through a real stuck-type fault simulator which is obviously very convincing. The estimated length of test sequences is $O(n_R^2 + n_I^2)$ where $n_R$ and $n_I$ are the number of registers and instructions respectively.

To improve the computational efficiency, [8] proposed a simplified fault model with revised test sequence length of $O(n_R * n_I)$. The major shortcoming of this approach is that it does not include partial execution of instruction and control faults which can be provided by the RTL model.

The benefits of the RTL technique are its generality, clarity, and easy understandability and precise description of the behavior of a given digital system. Its major drawbacks are the difficulty in automatic generation of the RTL description from the available information about the chip-under-test, test generation, and the fault coverage of the tests. The RTL technique also presents particular problems in fault localization. The RTL sequence does not uniquely determine the way the microprocessor was implemented. Though RTL descriptions provide a comprehensive approach to define functional faults, they still are subjective to the problem of test complexity.

The microprogram-oriented approach is attractive because micro-instructions are basic to the operation of the control unit. For control functions, much saving in test length are possible for microprogrammed control paths. Like the RTL technique, the automatic generation of the proper microcodes is still an unsolved problem. The error coverage, though promising, remains to be proven. Also, the nature of the control path architecture may make fault localization pretty difficult [7,12].

## VII CURRENT TRENDS

Due to the small number of published research papers with solid results in this functional testing of LSI/VLSI devices field [6], and the increasing demands of such kinds of techniques from industries, functional testing is still one of the less-mature but most active areas in design automation (DA) development and the quality control (QC) process [19].

In this concluding section, we would like to point out briefly some important topics which are either under active investigation or require more work in the future.

As for the graph approach in [2], future improvements may include [2,8]:

1) Extending the architectural model to incorporate the features of the newer microprocessors.
2) Allowing for more general faults.
3) Generating tests for other microprocessors, e.g., bit-slice ones.
4) Design of a "compiler" to automatically transform the microprocessor description (e.g., at RTL level) into the test program.

The RTL techniques and microprogram-oriented technique need more work to obtain complete test procedures and automatic generation and verification of behavior description. Good techniques for measuring fault coverage are also needed in both approaches. Quantitative fault coverage measures, possibly by computer simulation, are attractive for the demonstration of the comprehensiveness of those approaches.

Furthermore, efficient algorithms should be developed for testing VLSI as well as digital systems containing mixed logic (e.g., mixture of VLSI, LSI MSI, and SSI integrated circuit chips). More work still needs to be done in using a hardware description language (HDL) to aid the generation of test at the functional level. A solid, comprehensive definition of fault coverage in functional level and its practical measure should also be developed to give criteria in judging the quality of the developed algorithms.

REFERENCES


1. R. Chantal, S. Gabriele, "Microprocessor Functional Testing," Digest of papers 1980 International Test Conference, pp. 433-443.
2. S.M. Thatte and J.A. Abraham, "Test Generation for Microprocessors," IEEE Transactions on Computers, Vol. C-29, No. 6, June 1980, pp. 429-441.
3. J.A. Abraham and S.M. Thatte, "Fault Coverage of Test Programs for Microprocessors," Digest of papers, 1979 International Test Conference, pp. 18-22.
4. Y. Min and S.Y.H. Su, "Testing Functional Faults in Digital Systems Described by Register Transfer Language," Journal of Digital Systems, Vol. 6, No. 2, Summer/Fall, 1982, pp. 161-183. Also, Digest of Papers, 1981 International Test Conference, pp. 447-457.
5. Y. Min and S.Y.H. Su, "Testing Functional Faults in VLSI," Proc. of the 19th Design Automation Conference, June 1982, pp. 384-392.
6. S.Y.H. Su, "Computer-aided Design and Testing of Digital Systems and Circuits," (Invited Paper), Proc. of the Jordan International Electrical & Electronic Engineering Conference, Amman, Jordan, April 25-28, 1983, pp. 75-82.
7. M.G. Lin, K. Rose, "Applying Test Theory to VLSI Testing," Digest of papers, 1982, International Test Conference, pp. 580-585.
8. J.A. Abraham and K.P. Parker, "Practical Microprocessor Testing: Open and Closed Loop Approaches," Digest of papers, 1981 International Test Conference.
9. R.L. Wadsack, "VLSI: How Much Fault Coverage is Enough?", Digest of papers, 1981 International Test Conference, pp. 547-554.

10. K. Son and J.Y.O. Fong. "Automatic Behavior Test Generation," 1982 International Test Conference, pp. 161-165.

11. J.Y.O. Fong, "Microprocessor Modelling for Logic Simulation," Digest of papers, 1981 International Test Conference, pp. 458-460.

12. M.A. Annaratone, M.G. Sami, "An Approach to Functional Testing of Microprocessors," Proc. 12th International Symposium on Fault-Tolerant Computing, Santa Monica, CA, June 1982, pp. 158-164.

13. Y.H. Levendal and P.R. Menon, "Test Generation Algorithms for Computer Hardware Description Language," 1982 IEEE Transactions on Computers, Vol. C-31, No. 7, pp. 577-588.

14. P.B. Banerjee and J.A. Abraham, "Fault Characterization of VLSI MOS Circuit," Digest of papers, 1982 International Test Conference, pp. 564-568.

15. M.A. Bruer, A.D. Friedman, "Functional Level Primitives in Test Generation," 1980 IEEE Transactions on Computers, Vol. C-29, No. 3, pp. 223-235.

16. T. Sridhar and J.P. Hayes, "A Functional Approach to Testing Bit-Sliced Microprocessors," 1981 IEEE Transactions on Computers, Vol. C-30, No. 8, pp. 563-571.

17. R. Parthasarathy, S.M. Reddy and J.G. Kuhl, "A Testable Design of General Purpose Microprocessors," Digest of papers, 1982 International Test Conference, pp. 117-124.

18. M.A. Breuer, A.D. Friedman, "Diagnosis & Reliable Design of Digital Systems," Computer Science Press 1976, Section 3.5.

19. M.A. Breuer, A.D. Friedman, A. Iosupovicz, "A Survey of the State-of-The-Art of Design Automation," Computer (IEEE), October 1981, pp. 58-75.

# CHAPTER 2

## New Algorithms for Testing

## Instruction Decoding Function of Microprocessors

### I. INTRODUCTION

Recent advances in LSI/VLSI have resulted in an exponential growth of the number of logic components in an integrated circuit (IC) chip. One such complex circuit is microprocessor. To ensure the reliable operations of microprocessors, it is important to have these devices tested prior to their use. Several researchers [1-6] have proposed different test procedures for testing these devices. Some of the more important approaches are briefly outlined below.

Thatte and Abraham [1,2] have made a significant contribution by proposing a graph model for microprocessors and, on the basis of the graph model, they developed test sequences to test a microprocessor. However, we believe that this method does not make use of all the information that is available to a user, as will become evident in the subsequent sections of this report.

Min and Su [3] have further reduced the test lengths for a number of different classes of faults in microprocessors but they do not consider all faults in the instruction decoding function (control unit).

Annartone and Sami [4] have proposed a method which relies on the knowledge about micro-instructions associated with each instruction of a microprocessor. The major limitation of the method is that some instructions remain untested.

Parthasavathy, Reddy and Kuhl [5] proposed a testable design to make the control part of a microprocessor testable. Thus the major limitation of their method is that it is not applicable to existing off-the-shelf microprocessors.

In this report, we propose a method to test the instruction decoding function of a microprocessor. The method proposed here is a generalization of the approach taken by Thatte and Abraham [2] and results in considerable reduction in the size of test programs.

In Section II, we describe the key ideas used in deriving the algorithms stated in this report. In Section III, necessary notations are developed which are used in Section IV to derive and prove different algorithms for testing microprocessors. Section V discusses the complexity of different algorithms. We also consider the Intel 8080 microprocessor as an example and compare the complexity of different algorithms for testing the instruction decoding function of this microprocessor. In Section VI we present an algorithm to detect a new class of faults and discuss its complexity. In Section VII we briefly outline how a complete microprocessor can be tested.

## II. KEY IDEAS

In this section, we describe the basic ideas used in this report to derive the necessary algorithms. Our study of a number of papers in the area of microprocessor testing [1-7] indicates that often the existing schemes have not made use of all the information available to users. For example, almost all microprocessor manufacturers provide a reasonable amount of timing information about their products. In particular, information on instruction execution time (in terms of the number of clock cycles) for each instruction is available to the users. Such information for a microprocessor can also be measured using simple test equipment. In the development of our first algorithm, we divide all instructions into different sets using the information on instruction execution time. The new objective, then, is to test instructions in different sets. Thus a larger problem is solved using the "divide and conquer" strategy by solving several smaller subproblems.

For the development of the second algorithm, we make use of Read and Write signals. In this scheme, each instruction is associated with an ordered Read-Write Sequence. This association is then used to divide the set of all instructions into different partitions, and as before, this division can be used to our advantage to solve the larger problem of testing all instructions.

In the last algorithm (Algorithm 3), we make use of both the above information, i.e., the number of clock cycles and the Read-Write sequence associated with each instruction to further subdivide the instruction set and derive the necessary tests.

In the following section, we develop the notation which can be used to divide the set of all instructions into different sets. Thus the concept of instruction execution time in terms of the numbers of clock cycles and Read-Write sequences is formally defined and functional notation is used to obtain the partitions of the set of all instructions.

## III. ASSUMPTIONS AND NOTATION

In this section, we introduce the necessary notation and present the fault model. This work is a generalization of that of Thatte and Abraham [2] therefore, the notation chosen is akin to that work. It is evident from the previous work of different researchers [1-7], and also pointed out by Thatte [6], the most complex and time consuming task of testing a microprocessor is detecting instruction decoding faults. We, therefore, concentrate on reducing the complexity of the test-set for detecting faults in the instruction decoding function. The set of all instructions for a microprocessor is denoted by $I = \{I_1, I_2, \ldots, I_n\}$. The type of faults which are assumed to take place in instruction decoding function are as follows [2].

(i) $I_j/\phi$ Fault : In the execution of an instruction $I_j$, no activity takes place in the microprocessor.

(ii) $I_j/I_k$ Fault : In the execution of an instruction $I_j$, a different instruction $I_k$ is executed.

(iii) $I_j/I_j+I_k$ Fault : In the execution of an instruction $I_j$, two instructions $I_j$ and $I_k$ are executed and both instructions are executed to completion.

Note that to verify that a fault $I_j/I_k$ is not present we need only to execute $I_j$ and observe that $I_j$ is executed correctly. However, to assure that a microprocessor is free of fault $I_j/I_j+I_k$, we must verify the $I_j$ is executed correctly whereas $I_k$ is not executed at all. Thus it is evident that the most time consuming task is the detection of the third type of faults $I_j/I_j+I_k$, as in this case we need to consider all possible pairs [6].

Thus the order of complexity for the test program is proportional to $|I|^2$, where $|I|$ denotes the cardinality of set I. Since $|I|=n$, the order of complexity is $O(n^2)$. We shall show that the size of test programs can be reduced considerably by partitioning the instruction set.

The execution of a program involves the fetch and execute steps for an instruction and then the next instruction fetch cycle begins. Let $T(I_j)$ be the number of clock cycles required for executing an instruction $I_j$ (including its fetch phase) before the next instruction in the program is fetched. Notice that this information is supplied by the manufacturers of microprocessors [8-11]. Furthermore, $T(I_j)$ for every $I_j \in I$ can be measured for a given microprocessor. At this stage, the following remarks are in order.

Remark 1: $T(I_j)$ is not the total time for fetching and execution, because for certain instructions, the execution phase may overlap with the fetch phase of the next instruction. Therefore, we have defined $T(I_j)$ to be the time for executing instruction $I_j$ before the next instruction is fetched.

Remark 2: For certain instructions, $T(I_j)$ is a range instead of a unique integer. Two examples are MUL (multiplication) and DIV (division) instructions in Intel 8086 microprocessor. For simplicity, the following theorems and algorithms shall assume $T(I_j)$ to be a unique integer. It is conceptually simple and straightforward to extend our results to microprocessors for which $T(I_j)$ is a range.

Definition 1: $F_1$ is a function mapping I into the set of all integers, Z ($F_1$: I→ Z), i.e., $F_1(I_j) = T(I_j)$, $I_j \in I$.

Also, let $k_1 = \max \{T(I_j) / \forall I_j \in I\}$.

Definition 2: Partition the set I into subsets $I^1, I^2, \ldots I^k$ as follows:

$$I^i = \{I_j / I_j \in I \text{ and } T(I_j) = i\}$$

Denote $|I^i| = n_i, 1 \leq i \leq k_1$.

Note that some such partitions can be empty.

Remark 3: It is evident that $\bigcup_{i=1}^{k_1} I^i = I$

Remark 4: $I^i$ is a partition (non-overlapping subset of instructions) if $T(I_j)$ is a single integer associated with $I_j$. If $T(I_j)$ is a range, the above method may or may not result in partitions. However, with minor modifications, the results of this paper will still apply.

We also observe that a read-write sequence (R-W Sequence) is associated with every instruction. For example, in Intel 8080 [8] the instruction "MVI (Move Immediate) data, M" consists of the following R-W sequence:

(i) The instruction is fetched - Read Cycle (R)

(ii) The data is read from Memory-Read Cycle (R)

(iii) The date is written to the Memory-Write Cycle (W)

Thus an ordered R-W sequence, RRW, is associated with the instruction "MVI data, M." Clearly, this sequence can be determined for every instruction. Furthermore, it can be observed by monitoring the read/write control signals of a microprocessor for a given instruction under execution. Some microprocessors have even additional information, e.g., idle period which is not included here to keep the treatment general. In fact, our study suggests that often more than

simply read/write information is available for most microprocessors. This concept is formally defined below.

Definition 3: A set $S_{k_2}$ be a set of ordered R-W sequences of length no more than $k_2$, starting with R, written as $S_{k_2} = \{R(R,W)^i;\ 0 \le i < k_2\}$. For an element $\alpha \in S_{k_2}$ we denote the length of $\alpha$ as $\ell g(\alpha)$

Example 1: $S_1 = \{R\}$, $S_2 = \{R, RR, RW\}$,

$S_3 = \{R, RR, RW, RRR, RRW, RWR, RWW\}$, etc.

$\ell g(RR) = 2$, $\ell g(RRW) = 3$.

Definition 4: $F_2$ is a function mapping I into $S_{k_2}$ ($F_2: I \to S_{k_2}$). for some $k_2$, i.e.,

$F_2(I_j)$ = R-W sequence associated with $I_j$

$= R^{j_1} W^{j_2} R^{j_3} \ldots$

where $R^{j_1}$ is R repeated $j_1$ times. Clearly, $k_2$ is equal to the length of the longest R-W sequence for some instruction.

Definition 5: Partition set I into subsets as follows: let $S_{k_2}$ then

$I^{\alpha} = \{I_j / I_j \in I \text{ and } F_2(I_j) = \alpha\}$.

The number of such nonempty partitions is finite. In fact, an upper bound on the number of such partitions is $2^{k_2} - 1$.

Definition 6: Let $\alpha,\beta\in S_{k_2}$. Let $\alpha=RX_2,\ldots,X_{\ell_1}$, $\beta=RY_2,\ldots,Y_{\ell_2}$, where $X_i$, $Y_i\in\{R,W\}$ and $\ell_1\leq\ell_2$. We say

(i) $\alpha\leq\beta$ if $X_i=Y_i$ for $2\leq i\leq\ell_1$.

(ii) $\alpha=\beta$ if $\alpha\leq\beta$ and $\beta\leq\alpha$

Example 2: (a) For $\alpha=RR$, $\beta=RRW$, $\alpha\leq\beta$

(b) For $\alpha=RW$, $\beta=RWRR$ $\alpha\leq\beta$

(c) For $\alpha=RW$, $\beta=RRWR$ $\alpha\not\leq\beta$

Definition 7: Let $I^\alpha$ and $I^\beta$ be two partitions induced by $F_2$. $I^\alpha$ is covered by $I_\beta$, written as $I^\alpha\leq I^\beta$, if and only if $\alpha\leq\beta$.

## IV. THEOREMS AND ALGORITHMS

In this section, we state and prove some theorems and algorithms for testing faults in the instruction decoding function.

Lemma 1: If $I_j \in I^i$ and $I_k \in I^\ell$ then for

(i) $I_j/I_k$ fault $T(I_j/I_K) = \ell$

(ii) $I_j/I_j+I_k$ fault $T(I_j/I_J+I_k) = \max(i,\ell)$.

Proof:

(i) It is straightforward to see that if instruction $I_k$ is executed instead of $I_j$ then

$$T(I_j/I_k) = T(I_k) = \ell .$$

(ii) If a fault $I_j/I_j+I_k$ is present, then by our assumption on faults both $I_j$ and $I_k$ must execute to completion. Therefore

$$T(I_j/I_j+I_k) = \max(T(I_j),T(I_k))$$
$$= \max(i,\ell)$$

Corollary 1: If $I_j \in I^i, I_k \in I^\ell$ and $i \neq \ell$ then fault $I_j/I_k$ can be detected by executing $I_j$ alone and observing $T(I_j)$

Lemma 2: If $I_j \in I^i$, $I_k \in I^\ell$ and $i<\ell$ then fault $I_j/I_j+I_k$ can be detected by executing $I_j$ alone and observing $T(I_j)$.

Proof: Clearly in the presence of fault $I_j/I_j+I_k$ the observed value

$$T(I_j/I_j+I_k) = \max(i,\ell)$$
$$= \ell$$

Thus $T(I_j) \neq T(I_j/I_j+I_k)$.

Therefore, observed and expected values will be different in the presence of the fault.

Normally, to detect faults $I_j/I_j+I_k$ one would need to test for every pair $(I_j,I_k)$ [2,6]. However, the implication of the above lemma is that we need not consider all pairs of instruction while testing the instruction decoding function. Thus, in the following algorithm, we execute each instruction $I_j$ and determine $T(I_j)$ then in the succeeding step we test for faults $I_j/I_j+I_k$, etc., using the partitions obtained in the previous section.

In Step 2 of the following algorithm, we do not present procedures to generate subprograms to test for faults $I_j/\phi$, $I_j/I_k$ and $I_j/I_j+I_k$ for given $I_j$ and $I_k$. One could use the procedure by Thatte and Abraham [2] for generating subprograms.

Algorithm 1: Algorithm to Detect Faults in Instruction Decoding Function Using Function $F_1^*$.

Step 1: Execute an instruction $I_j$ and observe $T(I_j)$ for all $I_j \varepsilon I$.

Step 2: For i=1 to $k_1$ do
begin

For all $I_j, I_k \varepsilon I^i$

begin

(a) Test for fault $I_j/\phi$.

(b) Test for fault $I_j/I_k$

(c) Test for fault $I_j/I_j+I_k$

end

For $\ell$:=1 to i-1 do

begin

(d) For all $I_p \varepsilon I^\ell$ , test for fault $I_j/I_j+I_p$

end.

end.

Theorem 1: Algorithm 1 detects all faults in the instruction decoding function.

Proof: By Corollary 1 and Lemma 2, all faults of the following types are detected in Step 1.

(i) faults $I_j/I_k$ with $T(I_j) \neq T(I_k)$

and

(ii) faults $I_j/I_j+I_k$ with $T(I_j)<T(I_k)$.

Step 2(a) detects faults of type $I_j/\phi$.

Step 2(b) detects faults of type $I_j/I_k$ with $T(I_j)=T(I_k)$

Step 2(c) detects faults of type $I_j/I_j+I_k$ with $T(I_j)=T(I_k)$

Step 2(d) detects faults of type $I_j/I_j+I_k$ with $T(I_j)>T(I_k)$

Thus all faults are detected by Algorithm 1.

We know proceed to the development of second algorithm by making use of R-W sequences. The following lemmas are established before the algorithm is stated.

Lemma 3: If $I_j \varepsilon I^{\alpha}$ and $I_k \varepsilon I^{\beta}$ then for fault $I_j/I_k$ we have $F_2(I_j/I_k) = \beta$

Proof: Clearly, $F_2(I_j/I_k) = F_2(I_k) = \beta$

Corollary 2: If $I_j \varepsilon I^{\alpha}$ and $I_k \varepsilon I^{\beta}$ and $\alpha \neq \beta$ then fault $I_j/I_k$ can be detected by executing $I_j$ alone and observing $F_2(I_j)$.

Definition 8: Let $\alpha, \beta \in S_{k_2}$ and

$$\alpha = RX_2X_3,\ldots,X_{\ell_1}$$

$$\beta = RY_2Y_3,\ldots,Y_{\ell_2}$$

then $\alpha \cup \beta = R(X_2 \cup Y_2)(X_3 \cup Y_3),\ldots,(X_{\ell_1} \cup Y_{\ell_1})\ Y_{\ell_1+1},\ldots,Y_{\ell_2}$

where $\cup$ denotes set union.

Lemma 4: If $I_j \in I^\alpha$ and $I_k \in I^\beta$ then for fault $I_j/I_j+I_k$

$$F_2(I_j/I_j+I_k) = \alpha \cup \beta$$

Proof: In the presence of fault $I_j/I_j+I_k$, the observed R-W sequence will be a union of R-W sequences for $I_j$ and $I_k$. (Note: "Union" in this context does not mean logical OR. In fact it only means that when both R and W signals are put on the control line at the same instance, the associated control lines will have the identical logical value. This is different from the situation where only one of the two control signals is active.)

Corollary 3: If $I_j \in I^\alpha$ and $I_k \in I^\beta$ and $I^\alpha \leq I^\beta$ then for fault $I_j/I_j+I_k$

$$F_2(I_j/I_j+I_k) = \beta$$

Lemma 5: Let $I_j \in I^\alpha$ , $I_k \in I^\beta$ . A fault $I_j/I_j+I_k$ can be detected by executing $I_j$ alone and observing $F_2(I_j)$ provided either one of the following two conditions is satisfied.

(a) $\ell g(\alpha) < \ell g(\beta)$

(b) $\ell g(\alpha) \geq \ell g(\beta)$ and $I^\beta \not\leq I^\alpha$,

where $\ell g(\alpha)$ denotes the length of Read-Write sequence $\alpha$ .

Proof: (a) From Lemma 4, $F_2(I_j/I_j+I_k) = \alpha \cup \beta$

If $\ell g(\alpha) < \ell g(\beta)$ then $\ell g(\alpha \cup \beta) = \ell g(\beta)$

Therefore $\ell g(F_2(I_j/I_j+I_k)) = \ell g(\beta) \neq \ell g(F_2(I_j)) = \ell g(\alpha)$.

(b) $\Gamma^{\beta} \not\subseteq \Gamma^{\alpha}$ implies $\beta \not\subseteq \alpha$

$\beta \not\subseteq \alpha$ implies $\alpha \cup \beta \neq \alpha$.

Therefore $F_2(I_h/I_j+I_k) = \alpha \cup \beta$

$\neq \alpha$

Once again, the implication of the above result is that we need not consider all pairs of instructions while testing the instruction decoding function. Thus we obtain a second algorithm given below. As in Algorithm 1 we use the partitions obtained in the previous sections to detect different faults in the instruction decoding function.

Algorithm 2: Algorithm to Detect Faults in Instruction Decoding Function Using Function $F_2$:

Step 1: Execute an instruction $I_j$ and observe the Read-Write sequence associated with $I_j$, i.e., $F_2(I_j)$ for all $I_j \epsilon I$.

Step 2: For every $\alpha \epsilon S_{k_2}$ do

begin

For all $I_j, I_k \epsilon I^{\alpha}$

begin

(a) Test for fault $I_j/\phi$

(b) Test for fault $I_j/I_k$

(c) Test for fault $I_j/I_j+I_k$

end

For $\beta \leq \alpha$ do

begin

For all $I_p \epsilon I^{\beta}$ test for fault $I_j/I_j+I_p$

end

end.

Theorem 2: Algorithm 2 detects all faults in the instruction decoding function.

Proof: The proof is similar to the proof of Theorem 1.

We have stated two independent algorithms for testing the instruction decoding function of a microprocessor. It is felt that the above two algorithms, if merged, result in an algorithm which will perform better than either of the above two algorithms. We will not state the necessary lemmas but the basic idea is as follows.

Each subset $I^i$ can be further partitioned into smaller subsets by defining function $F_2$ from $I^i$ to $S_i$.

We use Algorithm 1 to test faults in $I^i$ and $I^j$ classes. To test instructions in $I^i$ for different faults, we can use Algorithm 2. Formal statement of the algorithm is as follows:

Algorithm 3: Algorithm to Detect Faults in Instruction Decoding Function Using Functions $F_1$ and $F_2$.

Step 1: Partition I into Subsets $\{I^i\}$ using function $F_1$.

Step 2: For i=1 to $k_1$ do

begin

(a) Partition $I^i$ into subsets $\{I^\alpha\}$ using function $F_2$ and use Algorithm 2 to detect faults in the instructions within each subpartition.

(b) Use Algorithm 1 to detect the remaining faults.

end.

The above algorithm can also be stated by interchanging the order of application of $F_1$ and $F_2$, i.e., in Step 1, $F_2$ can be used to partition I and then in Step 2, each partition obtained in Step 1 can be further subdivided using Function $F_1$.

## V. COMPLEXITY OF THE ALGORITHMS

In this section, we calculate the complexity of our algorithms and compare our results with a well-known algorithm [2,6]. As our algorithm relies on known methods of generating test subprograms, we first determine the length of such programs to compute the complexity of our algorithms.

Thatte and Abraham [2] have shown that the length of test programs is determined by the tests which detect faults $I_j/I_j+I_k$. They also give algorithms to detect these faults. In their procedure one needs to consider all possible pairs $(I_j,I_k)$. As the number of such pairs is in the order of $n^2$, denoted by $O(n^2)$, the test complexity is $O(n^2)$. Let us assume that the actual test size is $c.n^2$ where c is a constant of proportionality. As we propose to use their algorithm for generating test programs for faults $I_j/I_j+I_k$, we need not know the value of c to determine the relative performance of our algorithms.

The following theorems give the complexity of Algorithms 1 and 2.

Theorem 3: The complexity of Algorithm 1 is

$$O\left(\sum_{i=1}^{k_1}\ \sum_{q=1}^{i} |\Gamma^i|.|\Gamma^q|\right)$$

Proof: As stated above, the complexity is determined by the number of faults of type $I_j/I_j+I_k$ for which we need to test the pair $(I_j,I_k)$.

As is evident from Algorithm 1 and the associated lemmas, many such pairs need not be considered. Only those pairs $(I_j,I_k)$ for which $T(I_j)$ $T(I_k)$ need to be considered.

Using this argument it is a simple matter to obtain the required order of complexity.

Theorem 4: The complexity of Algorithm 2 is

$$O\Big(\sum_{\alpha \in S_{k_2}} \sum_{\beta \leq \alpha} |I^{\alpha}||I^{\beta}|\Big)$$

Proof: The proof is similar to the proof of Theorem 3.

A similar expression for the complexity of Algorithm 3 can be written. As such an expression warrants only complex notation, it is not included here. However, in the following example, we compute the complexity of all three algorithms and compare the result with the complexity of Algorithm in [2,6].

Example 3: Let us consider the Intel 8080 [8] microprocessor. For this processor $|I|=72$. In this processor, some of the instructions can be tested independently but to keep the task of comparison simple, we assume that we need to consider all $(I_j, I_k)$ pairs.

(i) Using the algorithms by Thatte and Abraham [2], the size of the test program

$$TATP = c.n^2$$
$$= c.(72)^2$$
$$= 5184\ c.$$

(ii) Algorithm 1 results into different partitions as follows:

Let us denote $|I^i|$ as $n_i$, we then obtain

$n_1=n_2=n_3=n_8=n_9=n_{14}=n_{15}=n_{17}=0$

$n_4=20$, $n_5=7$, $n_6=1$, $n_7=21$, $n_{10}=13$, $n_{11}=3$,

$n_{13}=2$, $n_{16}=4$, $n_{18}=1$.

$$\sum_{i=1}^{18} \sum_{q=1}^{i} n_i n_q = 3137$$

Therefore, the size of the test program by Algorithm 1 is

AL1TP = 3137 c.

(iii) Algorithm 2 results into different partitions as shown below.

| | R | RR | RW | RRR* | RRW* | RWW | RRRR | RRRW | RRRRR | RRRWW |
|---|---|---|---|---|---|---|---|---|---|---|
| $\|I^{\alpha}\|$ | 29 | 19 | 2 | 8 | 4 | 3 | 1 | 1 | 1 | 4 |

$$\sum_{\alpha} \sum_{\beta \leq \alpha} |I^{\alpha}||I^{\beta}| = 2989$$

Therefore, the size of the test program by Algorithm 2 is

AL2TP = 2989 c.

(iv) Algorithm 3 results in different partitions as shown in the following table.

| R-W Seq. \ Clocks | 4 | 5 | 6 | 7 | 10 | 11 | 13 | 16 | 18 |
|---|---|---|---|---|---|---|---|---|---|
| R | 20 | 7 | 1 | | 1 | | | | |
| RR | | | | 19 | | | | | |
| RW | | | | 2 | | | | | |
| RRR | | | | | 7 | 1 | | | |
| RRW | | | | | 4 | | | | |
| RWW | | | | | 1 | 2 | | | |
| RRRR | | | | | | | 1 | | |
| RRRW | | | | | | | 1 | | |
| RRRR | | | | | | | | 1 | |
| RRRWW | | | | | | | | 3 | 1 |

* IN and OUT instructions have been included in RRR and RRW respectively to keep the treatment simple.

The size of the test program by Algorithm 3

AL3TP = 2761 c.

Comparison of (ii), (iii) and (iv) with (i) shows that these algorithms reduce the test program size by about 40%.

We have also considered other microprocessors, e.g., Intel 8085 [8], 8086 [9], Motorola 6800 [10], Z80 [11] and found that our partitioning methods are applicable to these microprocessors and typical saving in the size of test program is of the above order.

## VI. SYMMETRIC FAULT MODEL

In previous sections, we have considered faults $I_j/I_j+I_k$ and $I_k/I_k+I_j$ to be two distinct faults. However, if some information about the design of the control unit is available, then some of the conditions on the fault model can be modified.

For example, consider the microprogrammed implementations of a control unit. In such an implementation of the control unit, a stuck-type fault will always cause the execution of some micro-instructions, but a bridging fault between two control signals will cause either both controls to be active or inactive. Such a fault model is defined below.

Definition 9: A fault $I_j/I_j+I_k$ is symmetric if the presence of fault $I_j/I_j+I_k$ implies the presence of fault $I_k/I_j+I_k$ and vice versa. The following lemma and theorems can be proved using similar arguments as in the previous two sections.

Lemma 6: If $I_j \epsilon I^i$, $I_k \epsilon I^\ell$ and $i \neq \ell$ then symmetric fault $I_j/I_j+I_k$ can be detected by executing $I_j$ and $I_k$ and observing $T(I_j)$ and $T(I_k)$.

Proof: If $i<\ell$ then execution of $I_j$ will detect the fault whereas if $i>\ell$ then execution of $I_k$ will detect such a fault.

In the case of symmetric faults, Algorithm 1 is modified by eliminating Step 2(d); we shall call it Algorithm 1M.

Theorem 5: Algorithm 1M detects all symmetric faults in instruction decoding function.

Theorem 6: The complexity of Algorithm 1M is

$$O\left(\sum_{i=1}^{k_1} |I^i|^2\right).$$

Analogous results can be obtained by using R-W sequences.

## VII. CONCLUSION

In the previous sections, we have described algorithms to test the instruction decoding function. To test a microprocessor completely, we can adopt the following strategy.

(a) Register decoding function can be tested by using methods proposed in [2] and [3]. To detect faults in registers, test procedures identical to testing semiconductors RAMs [12,13] can be used.

(b) For detecting faults in data paths procedures given in [2,3,6] can be used.

(c) ALU faults can be tested by complete tests derived for a given realization or by random test sequences.

(d) Instruction decoding function faults can be detected by the algorithms given in this paper, in conjunction with the algorithms given in [6].

The algorithms presented in this report are valid for a number of microprocessors we have considered [8-11]. In our treatment, we have used only the information which is common to all these microprocessors, thus for any given microprocessor the actual complexity of the tests is likely to be even less than the complexity determined in this report.

REFERENCES


1. S.M. Thatte and J.A. Abraham, "A Methodology for Functional Level Testing of Microprocessors," Proc of the 8th International Symposium Fault-Tolerant Computing, Toulouse, France, June 1978.

2. S.M. Thatte and J.A. Abraham, "Test Generation for Microprocessors," IEEE TRANS. COMP., VOL. C-29, NO. 6, JUNE 1980, PP. 429-441.

3. Y. Min and S.Y.H. Su, "Testing Functional Faults in VLSI," Proc. 19th Design Automation Conf, Las Vegas, NV, 1982, pp. 384-392.

4. M.A. Annartone and M.G. Sami, "An Approach to Functional Testing of Microprocessors," Proc. 12th International Symposium on Fault-Tolerant Computing, Santa Monica, CA, June 1982, pp. 158-164.

5. R. Parthasarathy, S.M. Reddy and J. Kuhl, "A Testable Design of General Purpose Microprocessors," Proc. 12th International Symposium on Fault-Tolerant Computing, Santa Monica, CA June 1982, pp. 117-124.

6. S.M. Thatte, "Test Generation for Microprocessors," Report R-842, Co-ordinated Science Lab., Univ of IL - Urbana, May 1979.

7. T. Sidhar and J.P. Hayes, "Testing Bit-Sliced Microprocessors," Proc. 9th International Symposium on Fault-Tolerant Computing, Madison, WI, June 1979, pp. 211-218.

8. MCA-80™Family Users Manual, Intel Corp. Pub., Oct 1979.

9. The 8086 Family Users Manual, Intel Corp. Pub., Oct 1979.

10. M6800 Microprocessor Application Manual, Motorola Inc., 1975.

11. Z80™CPU, Z80A™CPU Technical Manual, Silog. Inc., 1977.

12. D. S. Suk and S. M. Reddy, "Test Procedures for a Class of Pattern Sensitive Faults in Semiconductor Random Access Memories," IEEE Trans. Comp., Vol. C-29, No. 6, June 1980, pp. 419-429.

13. A. Tuszynski (Organizer) - Workshop notes on memory testing, 1982 International Test Conference, Philadelphia, PA, Nov 1982.

## APPENDIX

Report for Trip to Jordan and Israel, 1983.

Recently, I was invited to give a paper entitled "Computer-aided Design and Testing of Digital Systems and Circuits" at the Jordan International Electrical and Electronic Conference which was held in Amman, Jordan April 25-28, 1983. The talk was very well received and various good comments were made by the attendees from all over the world. In this talk, I presented the recent development and pointed out the future directions in the important area of computer-aided design and testing of digital systems. I also introduced our project sponsored by the Army, outlined the key ideas and obtained feedback from the audience. Professor Sergio Brofferio of Politechnic Di Milano, Milano, Italy is so interested in this project that he invited me to go to his institution to present in detail my research results and to discuss research with their research group. Professor Broferrio's speciality is in the VLSI area. We had a lot of discussions on the subject which enhances this Army research project. He is also interested in the applications of VLSI in the area of video conferences. His colleague, Professor M.G. Sami is working in the testing of VLSI using microprogrammed approach. One of his recent publications has been cited in this report. His work is pretty close to my project for the Army. I plan to discuss VLSI testing with him in Italy.

I have also interacted with the following professionals.

1) Dr. M. Maqusi, Dr. I. Zabalawi and Dr. M.K. Abdelazuz of the University of Jordan.
2) R.C.V. Macario, College of Swansea, Swansea, United Kingdom.
3) Professor K.R. Rao, University of Texas at Arlington.
4) Professor L.C. Ludeman, New Mexico State University.

Mr. S.Y. Duda of the University of Cairo is doing research on the applications of microprocessors. He hopes to come to SUNY-Binghamton for his graduate studies.

Since the scope of the conference was broad, it helped to enhance my technical knowledge in various parts of electrical engineering. Perhaps the idea of testing can be applied to other areas.

I was invited to visit the Computing Center of the University of Jordan and give them some advice regarding the new computers which they want to purchase.

I was invited to give a talk at the Technion-Israel Institute of Technology at Haifa, Israel. Technion is the finest University in Israel in the area of Computer Science and Computer Engineering. The talk was entitled "Testing of VLSI." The purpose of this talk was to report the research ideas and results of my Army project and obtain some comments and suggestions. I also had extensive discussions with Professor M. Yoeli and Z. Kohavi. The followi comments and suggestions were made from them and the audiences of my talk.

1) The idea of using RTL (Register Transfer Language) to describe the behavior of the LSI/VLSI chips and borrowing the existing methods for testing gate level networks to the RTL level is very interesting and has a lot of potential in helping solve the difficult problems of testing VLSI.

2) The enumeration of all faults may be complex. The equi classes for functional faults should be found first.

3) Since the RTL description is not unique, how do we know the RTL description corresponds to the actual circuit behavior?

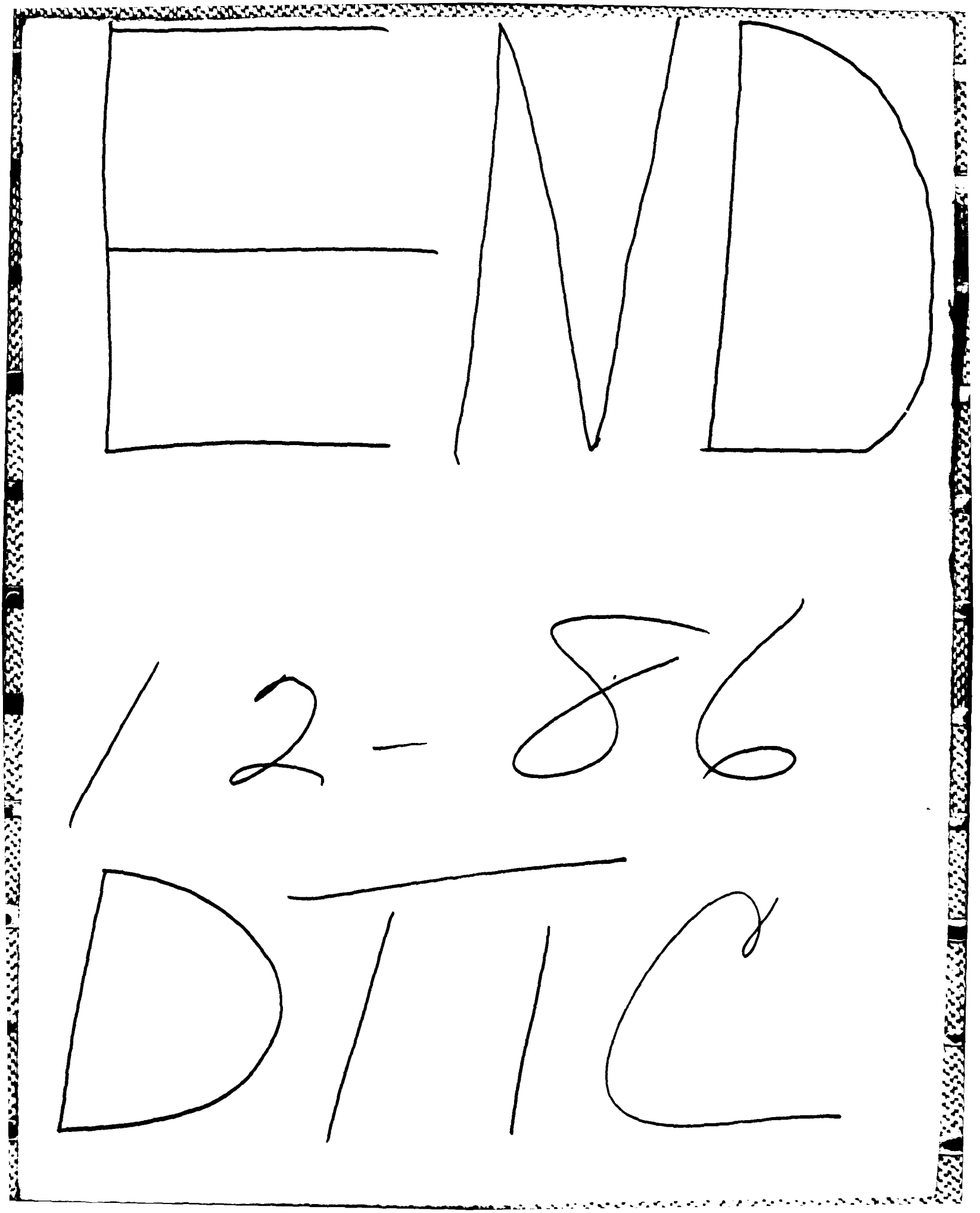
4) Further research should be done in the area of effectiveness of test generation algorithm, in terms of fault coverage and computing time.

5) Comparison between symbolic simulation and data graph approaches described in Su & Hsiech's paper "Testing Functional Faults in Digital System Described by Register Transfer Language" will be interesting and worthwhile.

6) The RTL description allows one to describe the behavior more closely than high level languages. The RTL model may be able to handle the fault for partial execution of an instruction which cannot be done by the existing techniques.

I have had a great deal of interaction with M. Baba of West Bank, who is interested in coming to SUNY-Binghamton to work on my Army research project. I also interviewed V.G. Habesh and Areej El-Majed, the other two applicants for this project.

Overall, the trip was very worthwhile and beneficial to the project. Suggestions were very helpful for the future progress on this project.


END
12-86
DTIC